

Japedo

User Manual

Version: 2.3



Table of Contents

Overview.....	1
Setup	2
Japedo Maven Plugin.....	2
Configuration properties	2
Logging	6
Representation and Navigation.....	7
JSON Format	7
PDF Format	7
HTML Format	7
Java Persistence Perspective	7
Database Perspective	12
Design Perspective.....	12
Graphical Representations	13
Linking database and JPA artefacts.....	14
Issue Report.....	15
Comparing databases and applications	16

Overview

Japedo is a tool for generating documentation of the complete persistence layer of a Java application. It creates documentation of the database scheme as classical database documentation tools do and in addition a full documentation of the JPA layer based on Java code analysis. If Java sources are available they will be scanned for persistence information and JavaDoc comments. This is the preferred mode to use Japedo as the JavaDoc represents a unique source of documentation for the persistence layer. If sources are not available the binaries are scanned but JavaDoc documentation will not be available. Brought together, it gives complete information about the persistent domain objects, their relations and persistent properties, their representation in the database scheme together with all properties of tables and columns. Together with descriptions of the persistent items the generated documentation is not only interesting for database admins but also for software developers, system analysts and other roles that need to understand the basic principles of the application.

Setup

This tool requires Java version 11 or higher.

NEW: Since version 2.3 Japedo is open source and can be downloaded from Maven Central with

```
<dependency>
  <groupId>com.logitags</groupId>
  <artifactId>japedo</artifactId>
  <version>2.3</version>
</dependency>
```

The distribution zip contains the Japedo jar library, the license, a configuration file and this user manual. Unzip into a directory and adapt the configuration file japedoConfig.xml as described below. The tool is started with:

```
java -cp japedo.jar;<path-to-database-driver> com.logitags.japedo.core.Main
```

The database driver can also be added to the dependencies property, see below, if this property is present (meaning if sources are analyzed together with the database)

Japedo Maven Plugin

If your persistence project is a Maven project, the easiest way to integrate Japedo is with the Japedo Maven plugin. This version of Japedo requires Japedo Maven plugin in version 1.3 or higher. Please see the [documentation](#) for the configuration.

Configuration properties

Most configuration properties have defaults and are optional.

Property	Description	Default
<code><version></code>	Version of the xsd file. Fixed to 2.0	
<code><properties><outputDirectory></code>	directory where Japedo will generate the documentation	current directory
<code><properties><outputFilename></code>	name of the generated html file in outputDirectory	japedoreport
<code><properties><outputFormat></code>	comma-separated list of output formats HTML, JSON, PDF, PDF-DB, PDF-JPA	HTML
<code><properties><logLevel></code>	log level (DEBUG , INFO , WARN , ERROR)	INFO
<code><properties><longDescriptionTarget></code>	If entity attribute descriptions don't fit into the table column, they can be displayed in a POPUP window or in the lower right PANEL .	POPUP

Property	Description	Default
<code><properties><proxyHost></code>	proxy host name (if any)	
<code><properties><proxyPort></code>	proxy port	
<code><properties><showAttributesOfHierarchy></code>	flag for indicating that the entity attribute table should display also inherited attributes from super classes.	false
<code><properties><showCounts></code>	flag to count and display number of entities, tables etc.	false
<code><properties><prettyPrint></code>	flag if the generated data files contain line breaks and indentations	false
<code><application name></code>	A name to provide a short identification for the application on which japedo is applied. The expression <code>\$2026-06-23T08:04:06Z</code> is resolved to the current date in format <code>YYYYMMdd</code> .	Japedo Persistence Documentation
<code><application><description></code>	A description of the application. Any optional comment or text. Use a CDATA element to include html tags	
<code><application><java><charset></code>	the charset to be used to parse the sources	UTF-8
<code><application><java><binaries></code>	<p>List the compiled binaries of the application here. The analysis of the sources requires a compilation, therefore the application and all dependencies of the application must be made available to Japedo. If the database is analyzed together with the sources, the driver binary can be listed here too. Dependencies could be provided in various formats as a semicolon-separated list:</p> <ul style="list-style-type: none"> • directories of .class files (exploded binaries) • .jar libraries of the compiled classes • .war files • a directory mixing the above 	

Property	Description	Default
<code><application><java><sources></code>	<p>the sources of the application as a semicolon-separated list. If not set, Japedo scans the binaries for persistence and entity information. If neither sources nor binaries are given, java analysis is skipped and only database documentation is generated. The source code can be provided in many formats:</p> <ul style="list-style-type: none"> • one or more .jar files • one or more .zip files • directories of .java files (exploded sources) • a directory mixing the above 	
<code><application><java><excludedSources></code>	<p>Entities might have properties of a type for which the source code is not available. If added in this property as a semicolon-separated list such types will be ignored. The qualified class name or a package name could be set. In the later case all classes of that package will be ignored.</p>	
<code><application><database><connectionUrl></code>	<p>The database connection URL. If not set database scheme analysis is skipped and only the source code is analyzed.</p>	
<code><application><database><driverClassName></code>	<p>database driver class name</p>	
<code><application><database><nbThreads></code>	<p>number of threads to use for database inspection (default is 4)</p>	
<code><application><database><password></code>	<p>database password</p>	
<code><application><database><schema></code>	<p>the schema name to analyze. Must be defined only when Japedo cannot detect it itself. Newer jdbc drivers can resolve the schema name. If the schema could not be detected and is not given as a configuration parameter an exception will be thrown.</p>	
<code><application><database><username></code>	<p>database user name</p>	
<code><application><database><sqlScript></code>	<p>path to an SQL script that will be executed before the database is analyzed. This allows for example to setup an in-memory database. INSERT, UPDATE and DELETE statements are ignored. If the path begins with classpath: the script is searched in the classpath. If the path begins with liquibase: a Liquibase changelog file is looked up in the filesystem or classpath and executed</p>	

Property	Description	Default
<code><application><ignoredIssues></code>	comma separated list of unique issue numbers that shall be ignored. This property can not be set when <code><application><dataFile></code> is set	
<code><application><dataFile></code>	a Japedo data file from a previous execution. It is also possible to define a directory. Then all files ending with <code>-app.js</code> are read	

An execution of Japedo can analyse multiple applications/databases at the same time. These are configured in element `<applications>`. An application must have a unique name and can be either configured by the `<java>` and/or `<database>` element or by the `<dataFile>` element, which defines the full path to the application file of a previous execution of Japedo. The `*-app.js` file is created in the output directory. In the html page that Japedo creates these applications can then be compared. This may be useful if you want to compare different installation environments or release versions of an application.

Logging

Japedo uses `java.util.logging` for logging messages during documentation generation. Japedo uses only log levels `DEBUG`, `INFO`, `WARN` and `ERROR` which can be set in `japedo.properties`. Of course also other levels from `java.util.logging.Level` could be configured. It is also possible to completely configure logging formats, handlers and other parameters by providing an own configuration class or logging properties file. These must be specified in system properties `java.util.logging.config.class` or `java.util.logging.config.file`. For more information consult [Java logging documentation](#).

Representation and Navigation

The documentation results are delivered as a dynamic html page, pdf or json, depending on the setting in the configuration file. Different formats can be generated at the same time.

JSON Format

A json file will be generated in the output directory with name applications.json. The structure and definitions of the json file are defined in a json schema file japedo-json-schema.json.

PDF Format

A PDF file will be generated in the output directory. Depending on the configuration in `<properties><outputFormat>` the generated pdf file has different content:

- PDF-DB: only database information is generated
- PDF-JPA: only Java JPA information is generated
- PDF: both database and Java information is generated

HTML Format

The main menu includes a menu item for configuring some general settings and a menu item for switching between the different applications if Japedo has been executed with multiple applications.

The results of database and Java persistence documentation are represented as one html page in the configured output directory. When displaying this page in a browser it will look similar to the representation of JavaDoc. On the left side are two windows with the general database and Java persistence artifact types in the upper and the special artifacts of the selected type in the lower window.

On the right side are two windows which display the details of a selected artifact. When selecting an artifact in the lower left window, the details will be displayed in the right windows.

Generally whenever clicking a link:

- mouse click opens link in upper right window
- Alt key + mouse click opens link in lower right window

Java Persistence Perspective

Java types include application, entities, embeddables, mapped super classes, enums and ID generators.

When clicking the Application item, content of all MANIFEST.MF files found in the source directory

and for the dependencies is displayed.

The lists of embeddables and mapped super classes can have duplicate entries because an embeddable could be embedded in more than one entity and mapped super classes can be the parent of more than one entity. Therefore it is mentioned, which entity is embedding an embeddable or is a child of a mapped super class.

Regarding the Enums, only classes that have relevance for persistence are displayed. The details of an enum are simple: it's just the enum description and the names, ordinal numbers and descriptions of the enum values.

The details of entities, embeddables and mapped super classes include the class hierarchy, the respective main table (or tables if secondary tables are defined) and the names, java types, descriptions, the mapped tables and column definitions and additional properties of the class attributes.

Columns [Name], [Type] and [Description] are self-explaining but columns [Properties] and [Table/Column] need some explanation. In order to fully understand the meanings of these columns a good understanding of the Java Persistence API is necessary.

The following attribute properties are listed optionally in column [Properties]:

Attribute property	Description
ID	This attribute is the unique id of the entity. Normally it is mapped to a primary key column in the database. An entity could have more than one ID attribute.
Generation type	<p>The mode how ID attributes and corresponding columns are set. The following types exist:</p> <ul style="list-style-type: none">• IDENTITY: Indicates that the value is set using a database identity column• SEQUENCE: Indicates that the value is set using a database sequence. A link to the sequence is provided• TABLE: Indicates that the value is set using an underlying database table to ensure uniqueness. A link to the table is provided• AUTO: Indicates that the persistence provider should pick an appropriate strategy for the particular database.• CODE: Indicates that the application is responsible to set a unique value.

Attribute property	Description
Replicated from super class	Indicates that an ID column is replicated from a super class when the inheritance strategy is JOINED and no discriminator column is defined. In this case, JPA expects a primary key column in the child class which contains the same value as the primary key column of the super class. The primary key column in the child class has no corresponding attribute in the entity.
Version	Specifies the attribute of an entity class that serves as its optimistic lock value. The version is used to ensure integrity when performing the merge operation and for optimistic concurrency control.
Enum as STRING/ORDINAL	Defines for attributes of type Enum if the corresponding column contains the name or the ordinal number of the Enum
Temporal as DATE/TIME/TIMESTAMP	Defines for attributes of type Date if the corresponding column contains date, time or timestamp values.
Discriminator values	Specifies the possible values of a discriminator column for the SINGLE_TABLE and JOINED inheritance strategies. The discriminator column has no corresponding attribute in the entity.
Relation	<p>Defines the relation type when the attribute type is a list or map. The following relation types are possible:</p> <ul style="list-style-type: none"> • n:m is a many to many relation • n:1 is a many to one relation • 1:n is a one to many relation • 1:1 is a one to one relation • List is a relation to a list of instances of a basic type or embeddable class. • Map is a relation to map of instances of a basic type, embeddable or entity class as keys and/or values
Mapped by	Specifies that the relation described in the [Table/Column] column is derived from the target entity specified in the [Type] column. The target entity is the owner of the relation.
Fetch type	Specifies the strategy for fetching data from the database, mostly relevant for relations. EAGER means the attribute values are loaded together with the parent entity. LAZY means the values are loaded only when they are first accessed in the application.

Attribute property	Description
Cascading	Defines the cascable operations that are propagated to the associated entity(s) in a relation-type attribute. Possible values are PERSIST , MERGE , REMOVE , REFRESH , DETACH
Not optional	Defines that the relation is not optional. A non-null relationship must always exist. If not specified, the relation is optional.

The [Table/Column] column contains the defined column name of an attribute. If the column is not in the main table of the entity, the table is additionally specified. A column may not be in the main table in three cases:

- The column is in a parent class of the entity
- A secondary table is defined for the entity and the column is in a secondary table.
- The attribute is a relation

While the first two cases are easy to understand, the third case needs more explanation:

1:1 relations

This type of relation could be mapped to different database implementations:

- **Join table:** The join table contains a column that references the owner entity of the relation and a column referencing the target entity. The [Table/Column] column specifies both these columns and the referencing columns in owner and target entities.
- **Foreign key:** The foreign key column is in the table of the relation owner. The [Table/Column] column specifies this column and which column it references in the target entity
- **Shared primary key:** The primary key of the owner entity is also the foreign key to the target entity. The [Table/Column] column specifies the same column name as the ID attribute referring to the primary key column of the target entity.

n:1 relations

In this kind of relations, the foreign key is on the owning side of the relationship which is normally the n side. The [Table/Column] column specifies the column representing the foreign key and which column it references in the target entity. Same as with a 1:1 relationship, a shared primary key could be used.

n:m relations

Many to many relations have always a join table containing a column that references the owner entity of the relation and a column referencing the target entity. The [Table/Column] column specifies both these columns and the referencing columns in owner and target entities. n:m relations with a map as type are special and described below.

1:n relations

Such relations could be implemented with

- A foreign key column in the table of the target entity. The [Table/Column] column specifies

the column representing the foreign key and which column it references in the table of the owning entity

- A join table where the [Table/Column] column specifies both the foreign key columns and the referencing columns in owner and target entities. 1:m relations with a map as type are special and described below.

List relations

List relations are lists of either a basic type or an Embeddable. Both cases are implemented using a join table. In the first case the [Table/Column] column specifies the column in the join table that refers to the primary key of the owning entity and the column that contains the basic type. When the list type is an Embeddable, the [Table/Column] column specifies only the column in the join table that refers to the primary key of the owning entity. When the attribute row is expanded by clicking the plus sign, the [Table/Column] column displays the column names for each attribute of the Embeddable.

Map relations

Relations that have in the [Type] column a map as Java type can be of type Map, 1:n or n:m. In the first case, the map value is of a basic type or an Embeddable. 1:n and n:m maps have an Entity as map value. The map key can in any case be of a basic type, an Embeddable, an Enum or an Entity. Such relations are implemented using a join table or by using foreign key columns in the table of the target entity. The [Table/Column] column for map relations specifies mostly three columns:

- The foreign key to the owner of the relationship referencing the primary key column of the owner entity
- A key column that holds the map key, labeled with 'Key'. This column could be a foreign key if the map key is an entity. If the map key is an embeddable, the [Table/Column] column lists the columns for all attributes of the embeddable as keys.
- A value column that holds the map value, labeled with 'Value'. This column could be a foreign key if the map value is a (target) entity. If the map value is a (target) entity and the relation is mapped to the entity table instead of a join table, the [Table/Column] column does not contain a value column.

Ordering

OneToMany, ManyToMany and element collection relations can specify an ordering column that is used to maintain the persistent order of a list. The persistence provider is responsible for maintaining the order upon retrieval and in the database. The order column has no corresponding attribute in the entity. In the [Table/Column] column an optional order column is noted with label 'Ordered by'.

Persistence Units

The dependencies are scanned for persistence.xml files and the persistence units are extracted and listed under the JPA types. The details of a persistence unit includes all properties and a list of the effective jpa classes that are managed by this persistence unit. The set of persistence classes that are managed by a persistence unit is defined by using one or more of the following:

- Annotated persistence classes contained in the root of the persistence unit if the exclude-

unlisted-classes property is false (default is true). The source of these classes is marked with '(implicit)'

- One or more object/relational mapping XML files (<mapping-file> property)
- One or more jar files that will be searched for annotated classes (<jar-file> property)
- An explicit list of classes (<class> property)

The set of effective classes managed by the persistence unit is the union of these sources. In the details of a JPA type it is listed which persistence units manage this type.

Database Perspective

The database scheme perspective includes some general database properties, tables, views, their indexes and constraints and sequences. For some databases also partitioning information is analyzed and the actual partitions are listed. We distinguish three types of tables and views:

- **Domain Object Tables/Views:** map directly to a JPA entity or embeddable. The attributes of the entity or embeddable are mapped to the columns of the table. It is possible that multiple entities map to the same table, for example the members of an entity hierarchy with a single-table inheritance strategy. The mapping entities are listed under 'Mapping Entities'
- **Association Tables:** map to an association between JPA entities or embeddables that use a join table. The participating entities are listed under 'Associating Entities'
- **Utility Tables/Views:** do not map to a JPA entity. They serve other purposes like providing non-normalized views for optimized database queries or managing unique identifier generation.

For each table/view the columns, indexes and constraints are listed with their corresponding properties. The settings for Delete and Update Rule of foreign key constraints need maybe some explanation. They define what happens to the foreign key when the referenced primary key in the referenced table is deleted or updated.

- **NO ACTION:** Delete and Update not allowed. Error message is generated.
- **CASCADE:** Foreign key row is also deleted or updated
- **SET NULL:** Foreign key is set to null
- **SET DEFAULT:** Foreign key is set to its default value. The primary key in the referenced table should also have a default value for this option.

Design Perspective

Even if already about 20 years old, the concept of domain driven design (DDD) by Eric Evans is still popular in the world of microservices and modular components. Many applications are more or less designed following the principles of bounded contexts, entity and value objects and applying a ubiquitous language for naming models. The design perspective tries to give an idea of the application's design and architecture making use of principles of DDD though the application must not necessarily be built on those concepts. It is meant for designers and software architects who are interested in the overall conception of the business domain. Also for those who know the business, the requirements, the customers, the business rules and processes but lack the technical

background this perspective reveals valuable insight. In this category may fall for example project managers, project owners, business analysts and business delivery.

This perspective uses the following assumptions:

- JPA entities correspond to DDD entities and represent the business domain objects
- all entities in a Java package belong to the same domain or sub-domain. The opposite is not necessarily given: a domain could be defined by more than one package.

By default, the package name is used as the domain name. This could be changed either to give it a more speaking name or to combine several packages in the same domain: In the package, add a package-info.java file and add in the JavaDoc comment a line with

```
Domain: "your domain name"
```

The design perspective presents the entity objects in the sense of DDD. The emphasis lies on the non-technical view of the business domain that is presented by the persistence objects. Only a minimum of technical details is shown and the domain information concentrates on the important domain objects, their attributes, types, descriptions and the relations between them. Please note the following explanations:

- Only JPA entities and their corresponding tables present business domain objects
- The business domain object name corresponds to the JPA entity name in upper case.
- JPA embeddables are technical implementations and their attributes are presented flattened within the business domain objects
- JPA mapped super classes are as well technical implementations and are directly incorporated in the domain objects
- The attribute names correspond to the JPA attribute names in upper case.
- The attribute types are presented as pseudo-types in order to keep it simple and technology agnostic. The pseudo-types are: alphanumeric, numeric, date, time, timestamp, boolean, binary, list and map.
- Discriminator and inherited id columns are technical implementations and not displayed
- All other technical implementation details have been omitted. The business domain object representations have a direct link to the corresponding JPA entities and database tables where these details can be found.

Graphical Representations

Each detail view of a table or JPA types includes a graphical representation. For a table the tables related by a foreign key constraint are also displayed. For JPA types, the class hierarchy and the directly associated JPA entities and Enum types are displayed. Clicking on a table or entity title with mouse click or Alt key + mouse click opens the details view of the item.




The application and database overview diagrams include in addition the graphical representation of the complete persistence layer and database structure. These graphics can be zoomed and

panned. For the overview diagrams there is a button to display a rectangle which can be used to magnify parts of the diagram. The size and scaling of the rectangle can be modified and it can be dragged around with the mouse. The part of the diagram outlined by the rectangle can be cut (use the Alt key to display the cut part in the lower window). The cut diagram sections can then be downloaded. The following overall diagrams are shown:

- entity diagram: all JPA entities, embeddables and mapped super classes and their relations
- table diagram: all database tables and their relations through foreign key constraints
- domain diagram: displays all domains and the containing domain objects
- Entity/Table Mappings: combines entity and table diagrams and displays the linkage between them
- Domain object/Table diagram: combines domain objects and tables and displays the linkage between them
- Domain object/Entity diagram: combines domain objects and entities and displays the linkage between them

Linking database and JPA artefacts

The complete picture of application persistence is obtained when database properties are studied together with corresponding domain object and JPA properties. For this purpose, table columns can be directly linked with JPA entity and domain object attributes. The three types are distinguished by their colors:

-  link with JPA attribute
-  link with domain object attribute
-  link with database column

Linking database columns with corresponding entity attributes is the most complicated linkage because it is a 1 to 0..n relationship. A column can have zero, one or many links to entity attributes.

The default case is visible in the example application with table CIB_CONTROLLABLE: When clicking the link icon of a column, it is highlighted together with its corresponding entity attribute. This is the one-to-one relationship.

One example of a one-to-many relationship can be seen in table CIB_ARCHIVE: When clicking the link icon of column ARCHIVEID, two attributes of the Archive entity are highlighted. This is because the primary key of Archive is at the same time the foreign key to the Resource entity which is mandated by the @PrimaryKeyJoinColumn annotation.

Another example of a one-to-many relationship is in table BOOK2: When clicking the link on column ID, two lines in the Book2 entity are highlighted. The primary key column from table PUBLICATION is duplicated in table BOOK2 because the inheritance strategy is JOINED. In the entity, however the attribute is not duplicated.

There are many other situations, where one column is mapped to multiple entity attributes, for example:

- Parent – child relationships (column CIB_EVENTRESULT.PARENTRESULT_ID)
- Part of an composite primary key is a foreign key (annotation @MapsId) ((column DEPENDENT.EMP_ID)
- Multiple relations maintained in the same association table (column MANYTOMANY1_MANYTOMANY2.MANYTOMANY1_ID1)
- Object inheritance (CIB_REQUEST)

The last point needs some more explanations: When an entity hierarchy is mapped to a single table (InheritanceType = SINGLE_TABLE) the columns of this table can be mapped to the different entities in this hierarchy. When clicking the link icon, the entity must therefore be selected to which the column shall be mapped. In this case, a dialog pops up which lists the possible entities which map the column. The entity can also be selected before clicking the link icon from the table's list of mapped or associated entities.

Some columns do not have a corresponding entity attribute. This is the case for example for columns of utility tables like a table for generating sequences. Order columns of one-to-many relations (annotation @OrderColumn) also do not have a relation to an attribute.

The mapping from JPA attributes to the corresponding columns is a 1 to 1..n relationship. The normal mapping is that an attribute maps to one column. In case of an attribute that represents an entity association which is mapped in an association table, both foreign key columns are mapped to that attribute. When an entity defines an order column (annotation @OrderColumn) for a Collection-type attribute, the order column is also mapped to that attribute. Another example for a 1 to n mapping is for attributes of Map type. In this case the foreign key column, the map value column and the map key column are mapped to that attribute if they are all in the same table.

Issue Report

Japedo can detect errors, mismatches between Java implementation and database, non-compliances to the JPA specification and many other flaws and disregards of common persistence rules and conventions. These are listed in the issue report with CRITICAL, MAJOR and MINOR error level. BLOCKER issues prevent Japedo from execution and are listed already in the execution log. The issue message contains links to the Java class or database table that produced the issue. The details view of the listed issues contain a more detailed explanation and a possible solution to the problem.

Issues that will not be fixed or are false-positives can be ignored so that they do not appear anymore on the issue report. Because the Japedo documentation is a html page based on Javascript we cannot update or write directly on the user's machine to store ignored issues. Therefore a three-step approach is followed:

- if issues are checked for ignore and confirmed in the popup they will be ignored only for the current browser session
- if the ignored issue ids are copied in the popup and added to the file ignoredIssues.js in the Japedo output folder, they will be ignored for the current documentation even when the browser is restarted
- if the ignored issue ids are copied in the popup and added additionally into the ignoredIssues

property of the Japedo configuration file, they will be ignored also in a follow-up execution of Japedo documentation generation. Beware that on each generation a fresh `ignoredIssues.js` is generated.

Comparing databases and applications

If more than one application is configured in `japedoConfig.xml` or in the Maven plugin Japedo compares the different databases and Java applications. An application can be a specific deployment installation of an application, for example in production and test environment or a specific release version of an application. With the Japedo Maven plugin the application version can be configured and the version is downloaded from the distribution repository. It is also possible to configure only databases or only sources in a application, then only the databases or sources are compared.

After execution of the Japedo analysis, the compare functionality can be accessed from the main menu of the generated html page. The compare page displays differences for database and Java source separately. For database differences, the table and column/constraint/index where a difference has been detected are listed. If a table/column/constraint/index is present in one database scheme and not in the other the entry lists where it is present and where not. The column 'Modified property' is empty in this case. If a property of a table/column/constraint/index differs, the column 'Modified property' is set and the entry lists the different properties.

Similarly the differences in the Java sources are listed with information about classes and class attributes present or not and the differing values of class or attribute properties.